Unifize Backend Developer Assignment

Please don't spend more than 3 hours on the implementation – the goal is to test your ability to translate real e-commerce business rules into clean, maintainable code. We are NOT looking for completeness of edge cases, one happy flow path is sufficient.

Assume you're building a fashion ecommerce website. This e-com offers three specific types of discounts

- Brand-specific discounts (e.g., "Min 40% off on PUMA", this will be across categories.)
- Bank card offers (e.g., "10% instant discount on ICICI Bank cards")
- Category-specific deals (e.g., "Extra 10% off on T-shirts")
- Vouchers (eg 'SUPER69' for 69% off on any product)

Technical Implementation :

Build a discount service that handles ecommerce-style discount scenarios. Using the following data models as black boxes, you are expected to complete the core service interface mentioned below.

Data Models

~~~
```python
from dataclasses import dataclass
from typing import List, Optional, Dict
from datetime import datetime
from decimal import Decimal
from enum import Enum

class BrandTier(Enum):
   PREMIUM = "premium"
   REGULAR = "regular"
   BUDGET = "budget"

@dataclass
class Product:
   id: str
~~~

```
    brand: str
    brand_tier: BrandTier
    category: str
    base_price: Decimal
    current_price: Decimal  # After brand/category discount

@dataclass
class CartItem:
    product: Product
    quantity: int
    size: str

@dataclass
class PaymentInfo:
    method: str  # CARD, UPI, etc
    bank_name: Optional[str]
    card_type: Optional[str]  # CREDIT, DEBIT

@dataclass
class DiscountedPrice:
    original_price: Decimal
    final_price: Decimal
    applied_discounts: Dict[str, Decimal]  # discount_name -> amount
    message: str
```

<u>Core Service Interface</u>

```python
class DiscountService:
    async def calculate_cart_discounts(
        self,
        cart_items: List[CartItem],
        customer: CustomerProfile,
        payment_info: Optional[PaymentInfo] = None
    ) -> DiscountedPrice:
        """
        Calculate final price after applying discount logic:
        - First apply brand/category discounts
        - Then apply coupon codes
        - Then apply bank offers
```

~~~~python
        """
        pass

    async def validate_discount_code(
        self,
        code: str,
        cart_items: List[CartItem],
        customer: CustomerProfile
    ) -> bool:
        """
        Validate if a discount code can be applied.
        Handle Myntra-specific cases like:
        - Brand exclusions
        - Category restrictions
        - Customer tier requirements
        """
        pass
```
~~~~

Dummy Data Creation :

Model the following scenario in your data model. Store it in `fake_data.py`

1. Multiple Discount Scenario:
   - PUMA T-shirt with "Min 40% off"
   - Additional 10% off on T-shirts category
   - ICICI bank offer of 10% instant discount

Testing:

Write a test case to validate the logic using the dummy data.

Evaluation Metrics

1. Core Features:
   - Accurate discount calculations
   - Proper discount stacking order
   - Clear validation rules

   - Detailed error messages

2. Code Organization:
   - Separation of validation and calculation logic
   - Clean interface for adding new discount types
   - Type hints and documentation
   - Clean, maintainable, extensible, testable code

Submission Guidelines

- Document any assumptions and technical decisions made in a `README.md`
- Add any relevant diagrams for your design if you think they add value.
- Include instructions on how to run the codebase.
- Share a public github link of the code.